Конспект лекции
Основы языка моделирования UML

Цель и задачи лекции

Цель – изучить основы языка моделирования UML.

Задачи:

1. Узнать, что такое UML и определить его назначение
2. Ознакомиться с основными конструкциями UML

План занятия

1. Определение языка моделирования UML и его история
2. Основные диаграммы UML

Определение

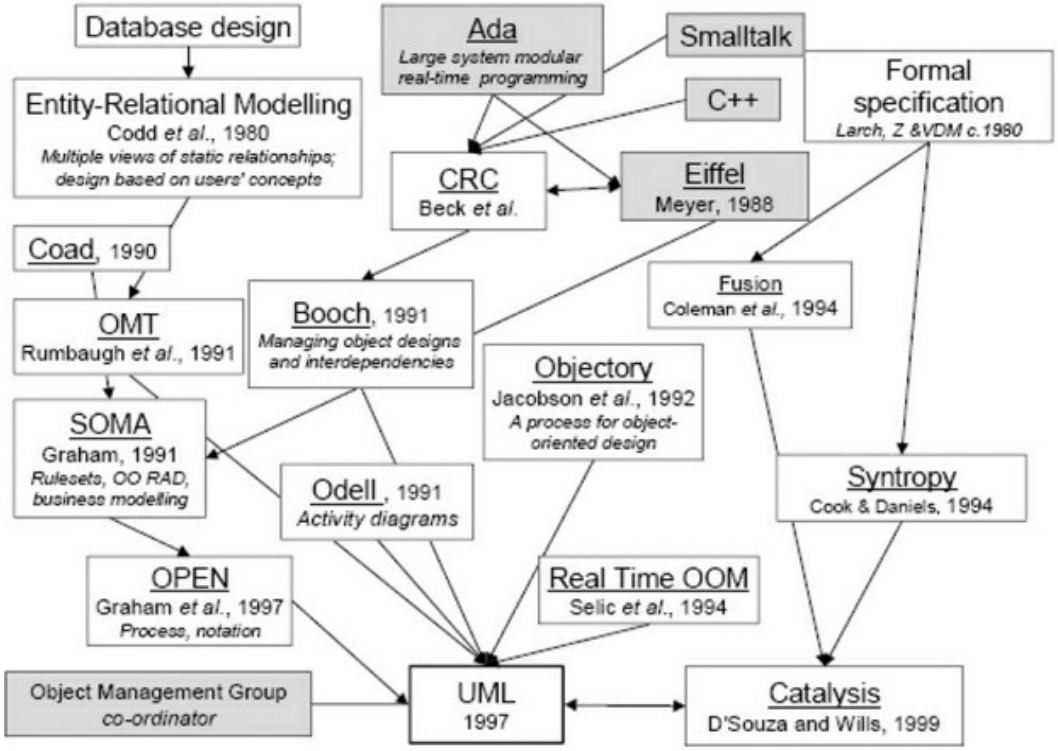
UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения,для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

Историческая справка

Откуда взялся The UML? Если говорить коротко, то UML вобрал в себя черты нотаций Грейди Буча (Grady Booch), Джима Румбаха (Jim Rumbaugh), Айвара Якобсона (Ivar Jacobson) и многих других.

В не такие уж и далекие 80-е годы было множество различных методологий моделирования. Каждая из них имела свои достоинства и недостатки, а также свою нотацию. То смутное время получило название "войны методов". Проблема в том, что разные люди использовали разные нотации, и для того чтобы понять, что описывает та или иная диаграмма, зачастую требовался "переводчик". Один и тот же символ мог означать в разных нотациях абсолютно разные вещи! На рисунке ниже можно увидеть лишь малую часть многообразия методов, которые существовали в то время и в какой-то мере повлияли на UML.



К тому же примерно в это же время (начало 80-х) стартовала "объектно-ориентированная эра". Все началось с появлением семейства языков программирования SmallTalk, которые применяли некоторые понятия языка Simula-67, использовавшегося в 60-х годах. Появление объектно-ориентированного подхода в первую очередь было обусловлено увеличением сложности задач. Объектно-ориентированный подход внес достаточно радикальные изменения в сами принципы создания и функционирования программ, но, в то же время, позволил существенно повысить производительность труда программистов, по-иному взглянуть на проблемы и методы их решения, сделать программы более компактными и легко расширяемыми.

Но самое главное, что появление ООП требовало удобного инструмента для моделирования, единой нотации для описания сложных программных систем. И вот "три амиго", три крупнейших специалиста, три автора наиболее популярных методов решили объединить свои разработки. В 1991-м каждый из "трех амиго" начал с написания книги, в которой изложил свой метод ООАП. Каждая методология была по-своему хороша, но каждая имела и недостатки. Так, метод Буча был хорош в проектировании, но слабоват в анализе. OMT Румбаха был, наоборот, отличным средством анализа, но плох в проектировании. И наконец, Objectory Якобсона был действительно хорош с точки зрения user experience, на который ни метод Буча, ни OMT не обращали особого внимания. Основной идеей Objectory было то, что анализ должен начинаться с прецедентов, а не с диаграммы классов, которые должны быть производными от них.

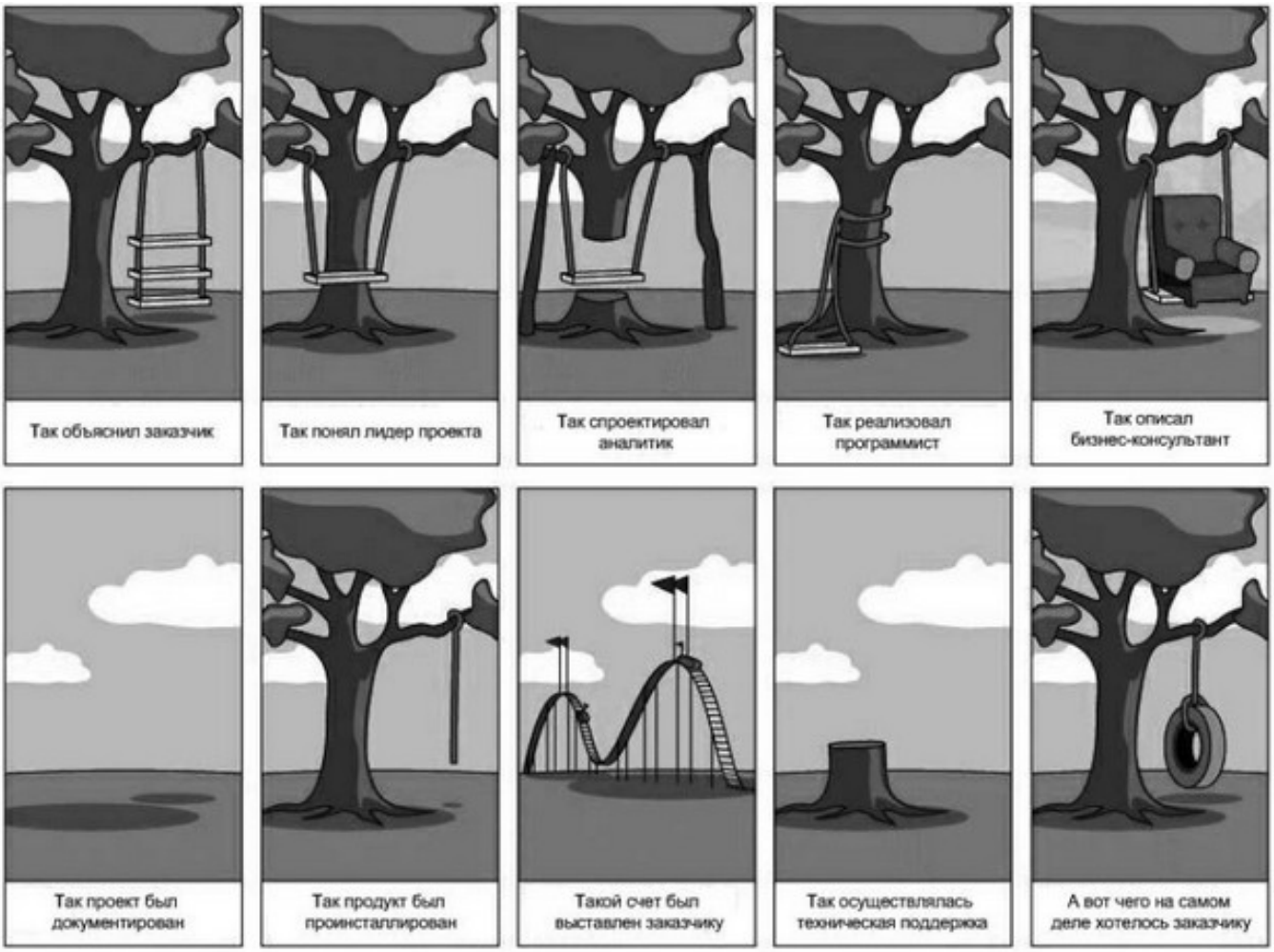
К 1994-му существовало 72 метода, или частные методики. Многие из них "перекрывались", т. е. использовали похожие идеи, нотации и т. д. Как уже говорилось выше, чувствовалась острая потребность, "социальный заказ"

- закончить "войну методов" и объединить в одном унифицированном средстве все лучшее, что было создано в области моделирования.

А что сейчас? The UML живет и развивается. Сейчас мы имеем UML 2.0 и десятки CASE-средств, поддерживающих UML. Вопреки популярному мнению, в наши дни Rational не владеет UML, но продолжает работать над ним. UML же принадлежит OMG, а сама Rational ныне является одним из подразделений IBM и фигурирует во всех документах как IBM Rational. UML же получил множество пакетов расширений, называемых профайлами и позволяющих использовать его для моделирования систем из специфических предметных областей.

Назначение языка UML

Начать хотелось бы с демонстрации известной картинки, которая уже более двух десятилетий "живет" в Интернете, но источник ее никому не известен. Эта картинка прекрасно иллюстрирует типичный процесс создания продукта.



Здесь мы видим все проблемы программной инженерии, в частности проблемы с коммуникацией и пониманием, вызванные отсутствием четкой спецификации создаваемого продукта. Так вот, авторы UML определяют его как графический язык моделирования общего назначения (т. е. его можно применять для проектирования чего угодно - от простых качелей, как на

рисунке, до сложного аппаратно-программного комплекса или даже космического корабля), предназначенный для спецификации, визуализации, проектирования и документирования всех артефактов, создаваемых в ходе разработки.

Виды диаграмм

UML 1.5 определял двенадцать типов диаграмм, разделенных на три группы:

- четыре типа диаграмм представляют статическую структуру приложения;
- пять представляют поведенческие аспекты системы;
- три представляют физические аспекты функционирования системы (диаграммы реализации).

Текущая версия UML 2.1 внесла не слишком много изменений. Диаграммы слегка изменились внешне (появились фреймы и другие визуальные улучшения), немного усовершенствовалась нотация, некоторые диаграммы получили новые наименования.

Впрочем, точное число канонических диаграмм для нас абсолютно не важно, так как мы рассмотрим не все из них, а лишь некоторые - по той причине, что количество типов диаграмм для конкретной модели конкретного приложения не является строго фиксированным. Для простых приложений нет необходимости строить все без исключения диаграммы. Например, для локального приложения не обязательно строить диаграмму развертывания. Важно понимать, что перечень диаграмм зависит от специфики разрабатываемого проекта и определяется самим разработчиком.

Итак, мы кратко рассмотрим такие виды диаграмм, как:

- диаграмма прецедентов;
- диаграмма классов;
- диаграмма объектов;
- диаграмма последовательностей;
- диаграмма взаимодействия;
- диаграмма состояний;
- диаграмма активности;
- диаграмма развертывания.

Диаграмма прецедентов (use case diagram)

Любые (в том числе и программные) системы проектируются с учетом того, что в процессе своей работы они будут использоваться людьми и/или взаимодействовать с другими системами. Сущности, с которыми взаимодействует система в процессе своей работы, называются экторами, причем каждый эктор ожидает, что система будет вести себя строго определенным, предсказуемым образом.

Эктор (actor) - это множество логически связанных ролей, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема

или класс). Эктором может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Прецедент (use-case) - описание отдельного аспекта поведения системы с точки зрения пользователя.

Прецеденты обозначаются очень простым образом - в виде эллипса, внутри которого указано его название. Прецеденты и экторы соединяются с помощью линий. Часто на одном из концов линии изображают стрелку, причем направлена она к тому, у кого запрашивают сервис, другими словами, чьими услугами пользуются. Это простое объяснение иллюстрирует понимание прецедентов как сервисов, пропагандируемое компанией IBM.

Примеры use-case диаграмм:

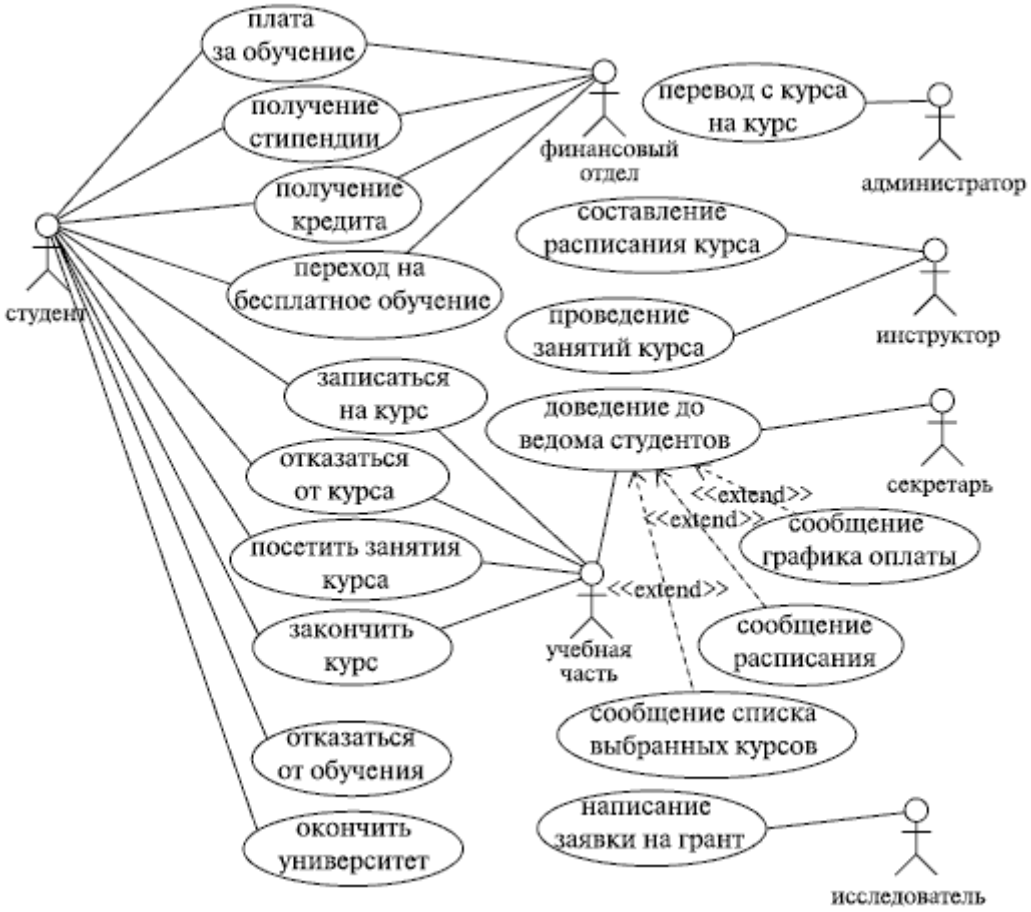


Диаграмма классов (class diagram)

Вообще-то, понятие класса нам уже знакомо, но, пожалуй, не лишним будет поговорить о классах еще раз. Классики о классах говорят очень просто и понятно:

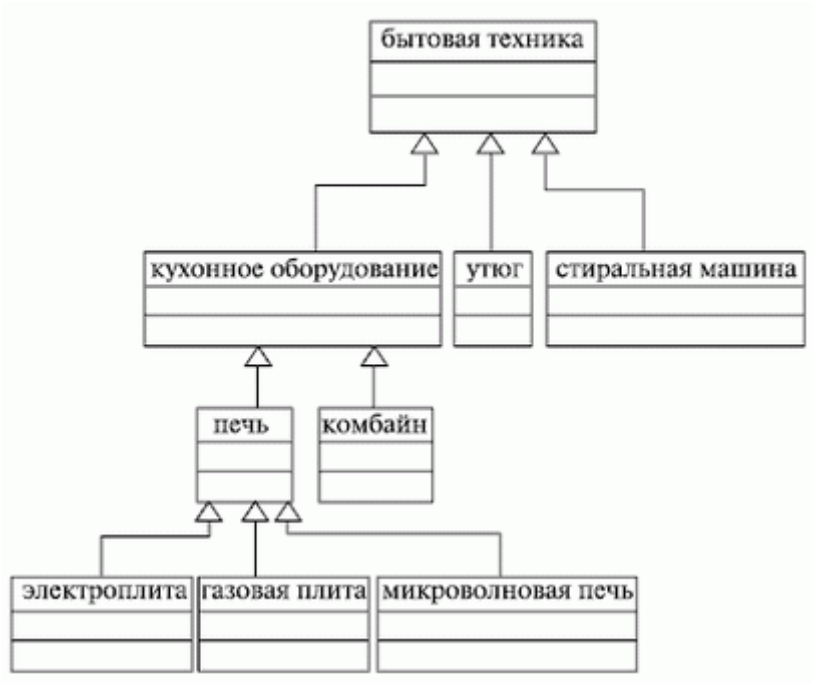
Класс (class) - категория вещей, которые имеют общие атрибуты и операции.

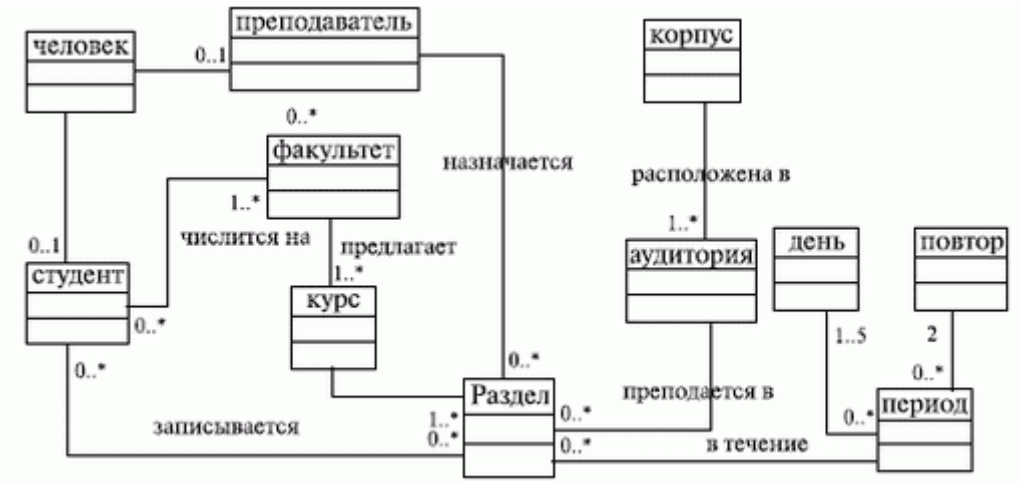
Продолжая тему, скажем, что классы - это строительные блоки любой объектно-ориентированной системы. Они представляют собой описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. При проектировании объектно-ориентированных систем диаграммы классов обязательны.

Классы используются в процессе анализа предметной области для составления словаря предметной области разрабатываемой системы. Это могут быть как абстрактные понятия предметной области, так и классы, на которые опирается разработка и которые описывают программные или аппаратные сущности.

Диаграмма классов - это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании - описании существующих и используемых систем. Информация с диаграммы классов напрямую отображается в исходный код приложения - в большинстве существующих инструментов UML-

моделирования возможна кодогенерация для определенного языка программирования (обычно Java или C++).





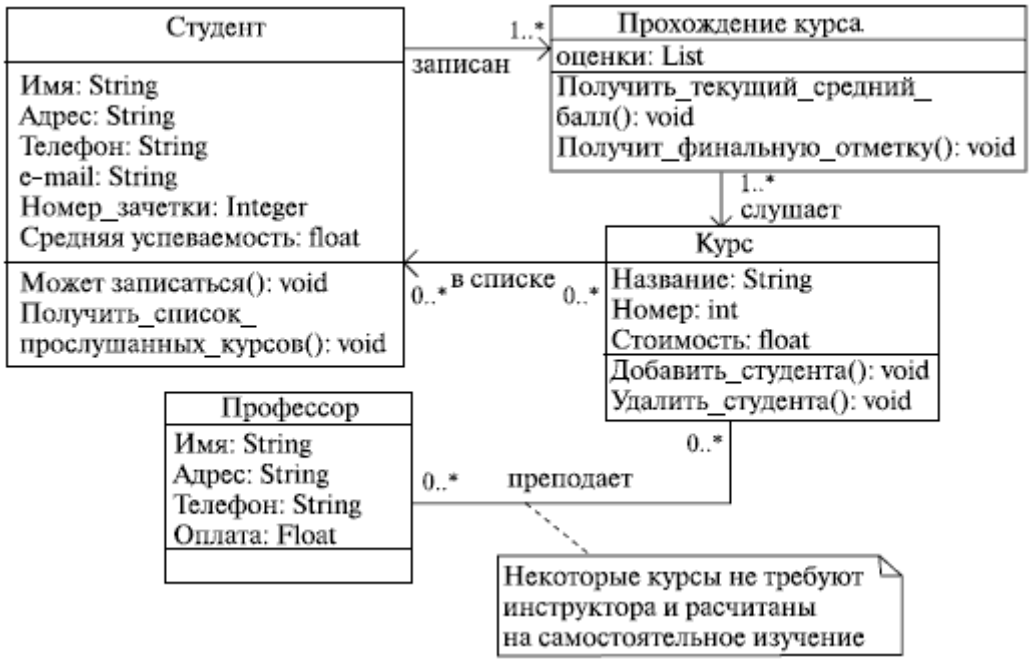
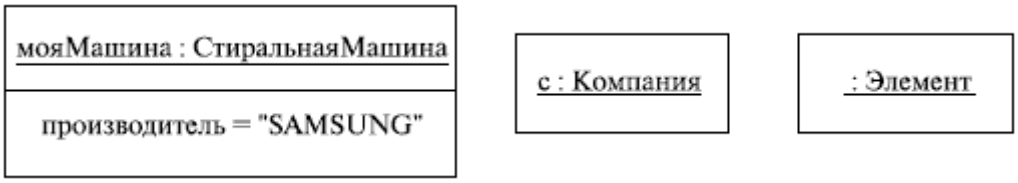


Диаграмма объектов (object diagram)

Объект (object) -

- конкретная материализация абстракции;
- сущность с хорошо определенными границами, в которой инкапсулированы состояние и поведение;
- экземпляр класса (вернее, классификатора - эктор, класс или интерфейс). Объект уникально идентифицируется значениями атрибутов, определяющими его состояние в данный момент времени.



Для чего нужны диаграммы объектов? Они показывают множество объектов - экземпляров классов (изображенных на диаграмме классов) и отношений между ними в некоторый момент времени. То есть диаграмма объектов - это своего рода снимок состояния системы в определенный момент времени, показывающий множество объектов, их состояния и отношения между ними в данный момент.

Таким образом, диаграммы объектов представляют статический вид системы с точки зрения проектирования и процессов, являясь основой для сценариев, описываемых диаграммами взаимодействия. Говоря другими словами, диаграмма объектов используется для пояснения и детализации диаграмм взаимодействия, например, диаграмм последовательностей.

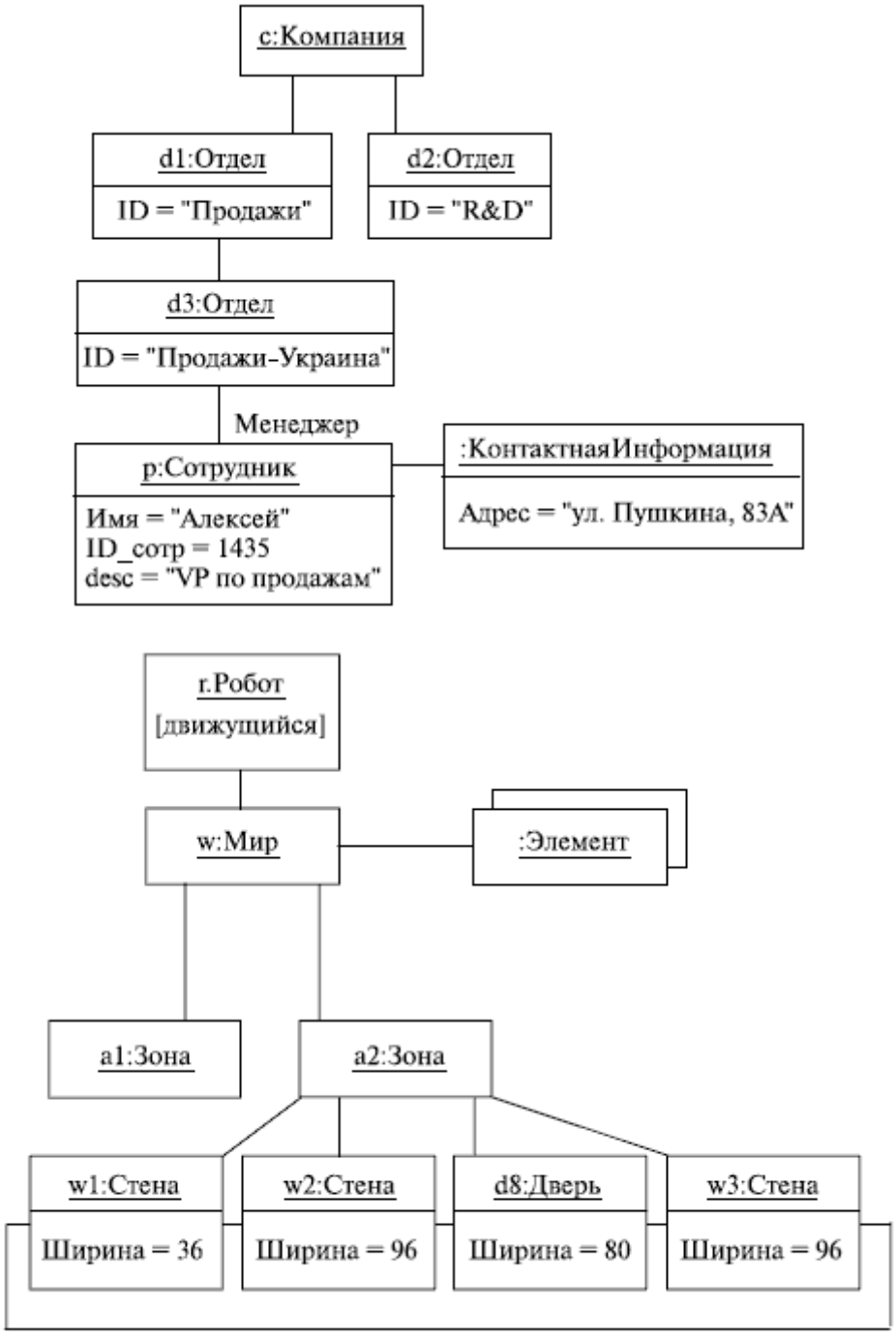


Диаграмма последовательности (sequence diagram)

Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов Системы во времени, а также обмена сообщениями между ними.

Одним из основных принципов ООП является способ информационного обмена между элементами Системы, выражающийся в отправке и получении сообщений друг от друга. Таким образом, основные понятия диаграммы последовательности связаны с понятием Объект и Сообщение.

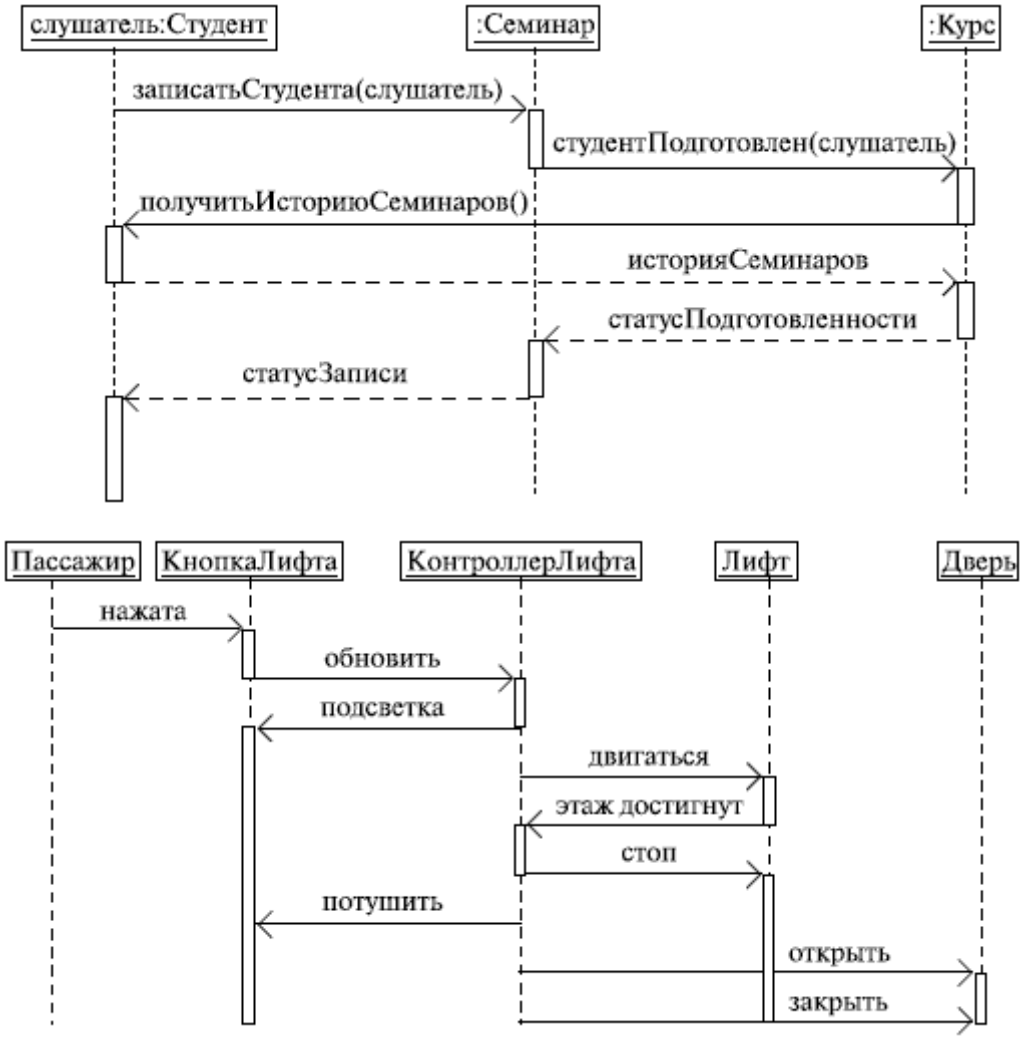


Диаграмма взаимодействия (кооперации, collaboration diagram)

Диаграмма взаимодействия показывает поток сообщений между объектами системы и основные ассоциации между ними и по сути, как уже было сказано выше, является альтернативой диаграммы последовательностей.

Следует отметить, что использование диаграммы последовательностей или диаграммы взаимодействия - личный выбор каждого проектировщика и зависит от индивидуального стиля проектирования. Мы, например, чаще отдаем предпочтение диаграмме последовательностей. На обозначениях, применяемых на диаграмме взаимодействия, думаем, не стоит останавливаться подробно. Здесь все стандартно: объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов, помните?), ассоциации между объектами указываются в виде соединяющих их линий, над ними может быть изображена стрелка с указанием названия сообщения и его порядкового номера.

Необходимость номера сообщения объясняется очень просто - в отличие от диаграммы последовательностей, время на диаграмме взаимодействия не показывается в виде отдельного измерения. Поэтому последовательность передачи сообщений можно указать только с помощью их нумерации. В этом

и состоит вероятная причина пренебрежения этим видом диаграмм многими проектировщиками.

Диаграмма состояний (statechart diagram)

Диаграмма состояний показывает, как объект переходит из одного состояния в другое. Очевидно, что диаграммы состояний служат для моделирования динамических аспектов системы (как и диаграммы последовательностей, кооперации, прецедентов и, как мы увидим далее, диаграммы деятельности). Часто можно услышать, что диаграмма состояний показывает автомат. Диаграмма состояний полезна при моделировании жизненного цикла объекта (как и ее частная разновидность - диаграмма деятельности, о которой мы будем говорить далее).

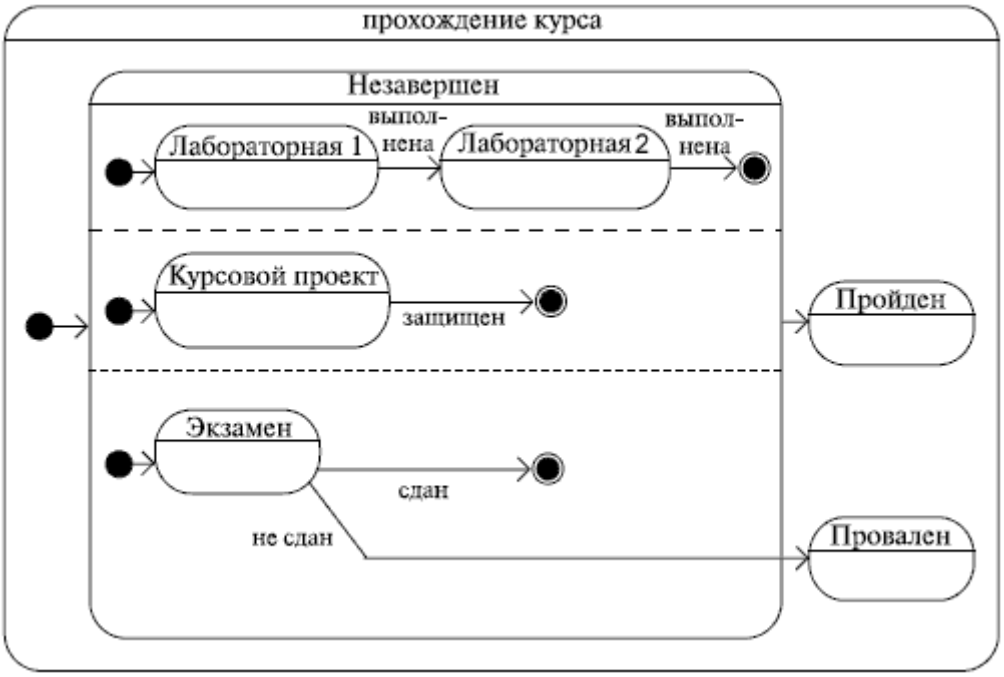


Диаграмма активности (деятельности, activity diagram)

Моделируя поведение проектируемой системы, часто недостаточно изобразить процесс смены ее состояний, а нужно также раскрыть детали алгоритмической реализации операций, выполняемых системой. В UML для

этого существуют диаграммы деятельности, являющиеся частным случаем диаграмм состояний. Диаграммы деятельности удобно применять для визуализации алгоритмов, по которым работают операции классов.

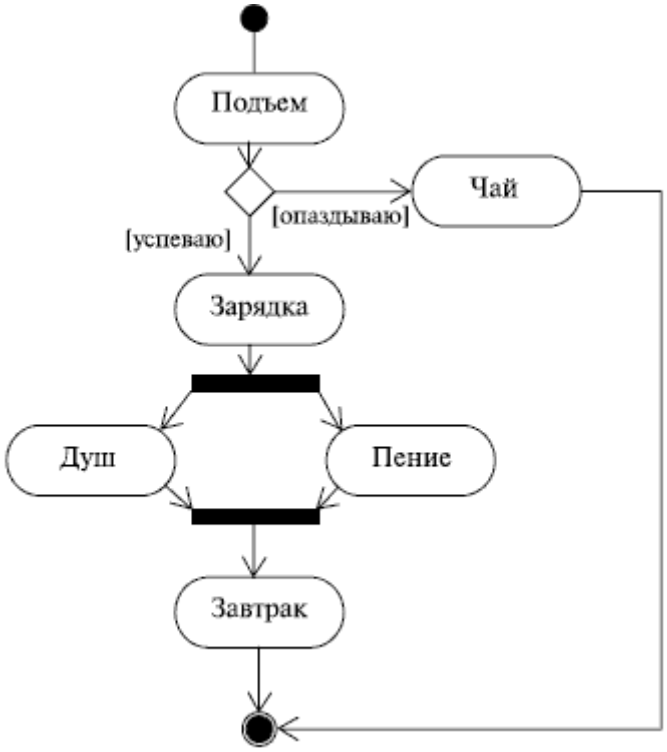
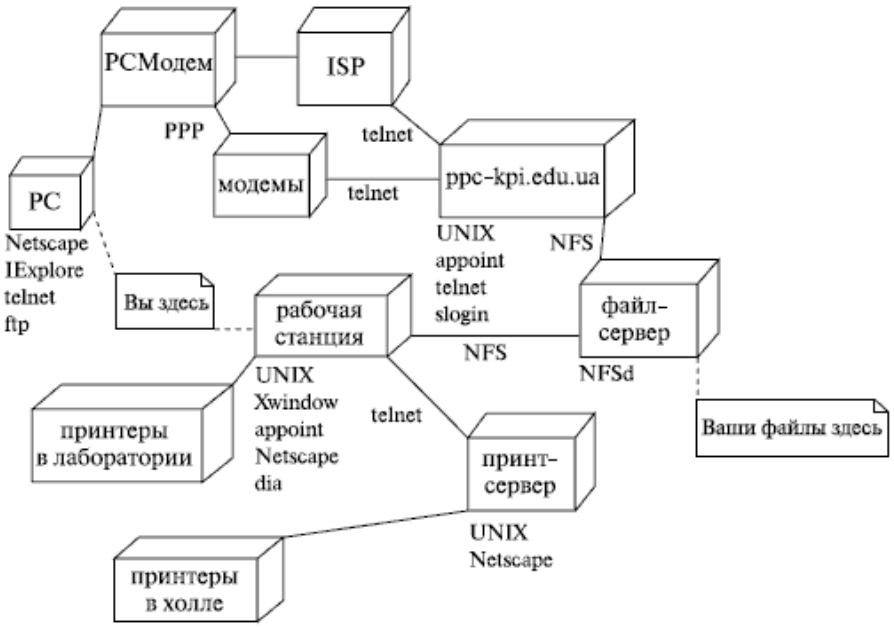


Диаграмма развертывания (deployment diagram)

Диаграмма развертывания показывает топологию системы и распределение компонентов системы по ее узлам, а также соединения - маршруты передачи информации между аппаратными узлами. Это единственная диаграмма, на которой применяются "трехмерные" обозначения: узлы системы обозначаются кубиками. Все остальные обозначения в UML - плоские фигуры.



Литература и ссылки

1. https://www.intuit.ru/studies/courses/1007/229/info
2. http://www.uml.org/
3. http://it-gost.ru/content/blogcategory/21/51/

Вопросы для самоконтроля

1. Как расшифровывается аббревиатура UML?
2. Кто были авторами UML?
3. Какие диаграммы были рассмотрены? Какое назначение каждого из них?
4. Почему нужно строить разные диаграммы при моделировании системы?